AI 응용시스템의 이해와 구축

10강. Serving

출석

Announcement

특강:

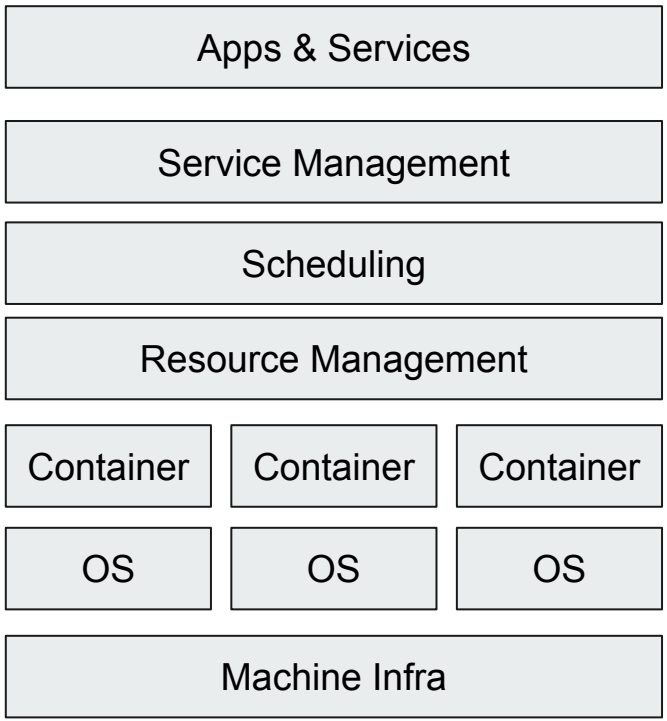
5/21 “Machine Learning in Audio Engineering” 이정석 박사 @ 메타 (Facebook)

Project 제출 + 발표

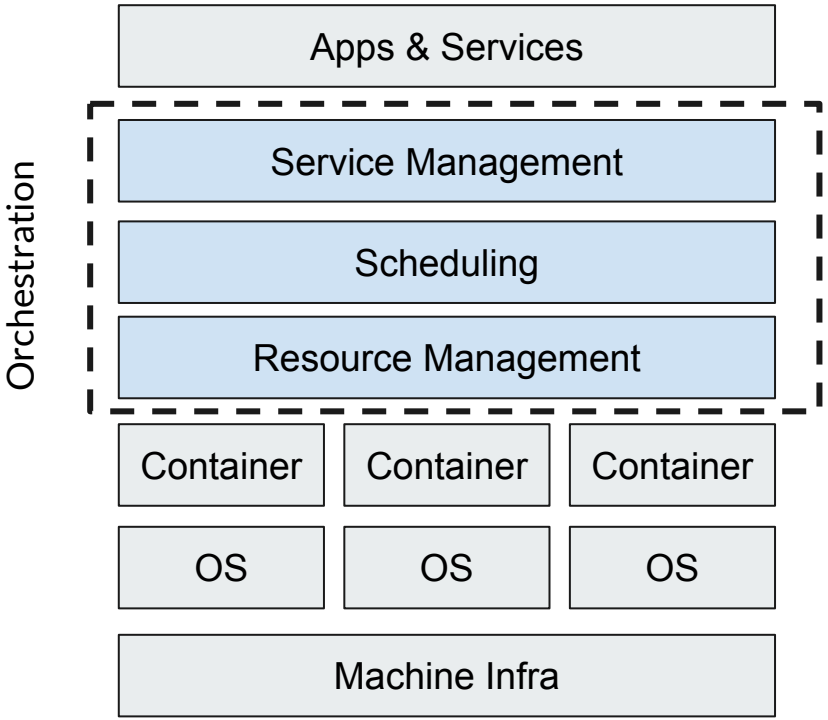
프로젝트	팀원	디자인닥 제출	발표
감정분석	권용순, 조우성, 박대혁	6/4	6/11
News Clip	박주형, 정운국, 김용욱	6/4	6/11
초해상화	정태훈, 안혜영, 조대선	6/4	6/11
상품 추천	노용문, 박준호, 한진웅, 이용정 (4 people team)	5/28	6/4
제조공정 불량 판정	임향빈, 오동규, 신용주, 정근시 (4 people team)	5/28	6/4

Serving

ML Serving Infrastructure



ML Serving Infrastructure



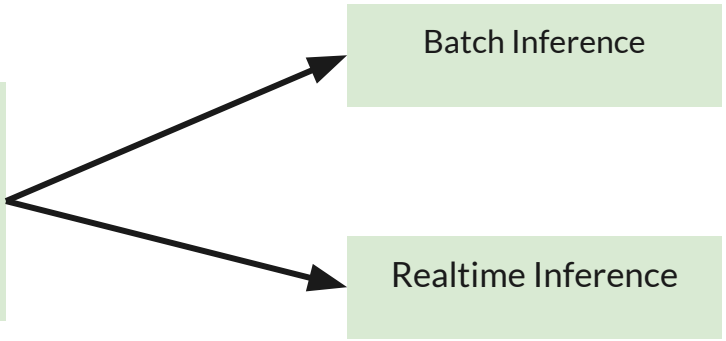
- 프로덕션 환경에서 콘테이너의 라이프사이클 관리
- 콘테이너의 스케일링 (scalability)
- 콘테이너의 신뢰성 (reliability)
- 콘테이너들 간의 리소스 분배
- 콘테이너 헬스 모니터링
- ...

Model Serving Pattern

- 모델
- 인터프리터 (Tensorflow interpreter)
- 인풋 데이터

Model Serving Pattern

- 모델
- 인터프리터 (Tensorflow interpreter)
- 인풋 데이터

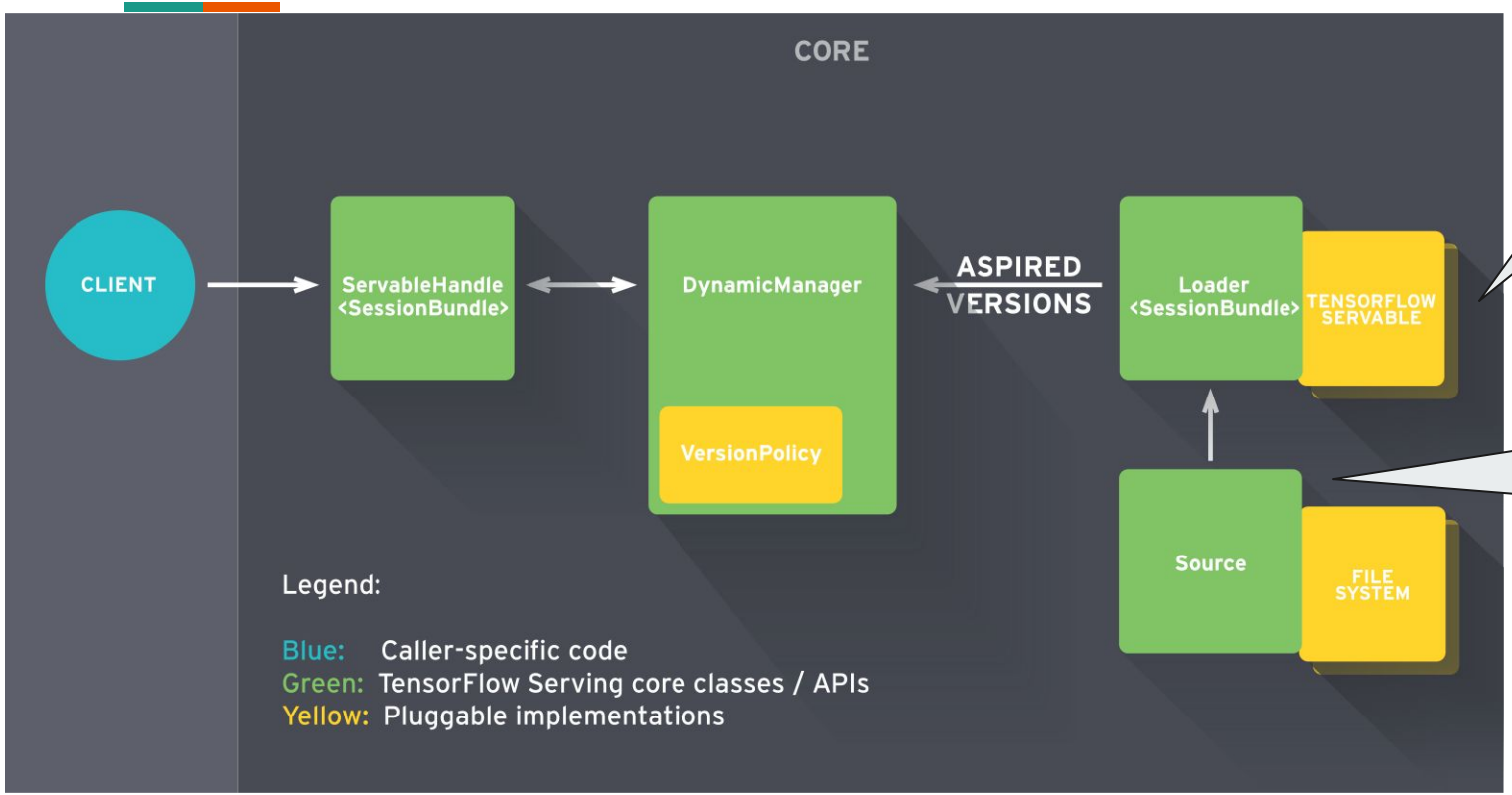


TensorFlow Serving

TF Serving: 프로덕션 머신러닝 셋업을 위한 서빙 시스템.

- Server API: Load TF model
- Client API: Request를 위한 REST API + gRPC API 제공
- Version Manager 제공.

TF Serving Architecture



Servable:
모델 번들.
Loader:
모델의
lifecycle 관리

Source:
servable을
내장.

Why use TF Serving “Managed Service”?

Low latency를 가진 실시간 prediction end point

학습된 모델을 직접 호스팅 (on premise) 에 디플로이하거나, GCP에 디플로이

트래픽에 따라 자동으로 스케일 (Scalability)

GPU / TPU를 사용한 prediction speed

Install TensorFlow Serving

This is all you need - one command line!

```
!{SUDO_IF_NEEDED} apt-get install tensorflow-model-server
```

Start running TensorFlow Serving

This is where we start running TensorFlow Serving and load our model. After it loads we can start making inference requests using REST. There are some important parameters:

- `rest_api_port`: The port that you'll use for REST requests.
- `model_name`: You'll use this in the URL of REST requests. It can be anything.
- `model_base_path`: This is the path to the directory where you've saved your model.

```
os.environ["MODEL_DIR"] = MODEL_DIR
```

```
%%bash --bg
nohup tensorflow_model_server \
  --rest_api_port=8501 \
  --model_name=fashion_model \
  --model_base_path="${MODEL_DIR}" >server.log 2>&1
```

```
!tail server.log
```

Newest version of the servable

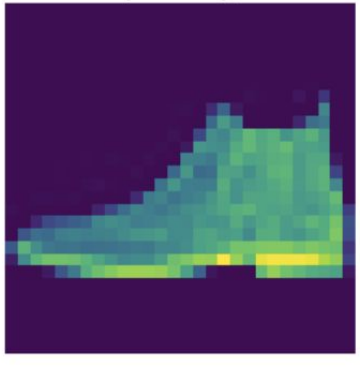
We'll send a predict request as a POST to our server's REST endpoint, and pass it three examples. We'll ask our server to give us the latest version of our servable by not specifying a particular version.

```
# docs_infra: no_execute
!pip install -q requests

import requests
headers = {"content-type": "application/json"}
json_response = requests.post('http://localhost:8501/v1/models/fashion_model:predict', data=data, headers=headers)
predictions = json.loads(json_response.text)['predictions']

show(0, 'The model thought this was a {} (class {}), and it was actually a {} (class {})'.format(
  class_names[np.argmax(predictions[0])], np.argmax(predictions[0]), class_names[test_labels[0]], test_labels[0]))
```

The model thought this was a Ankle boot (class 9), and it was actually a Ankle boot (class 9)



TF Serving Example

Colab example

Model serving infra

ML Infrastructure

- 학습 + 디플로이 전부 개인소유의 하드웨어 인프라에서 진행
- GPU/CPU 를 직접 서플라이 / 매니징
- 대형 시스템을 오랫동안 운영 시 적합 (대기업)

- 학습 + 디플로이를 클라우드에서 진행
- AWS, GCP, Azure, …

ML Infrastructure

오픈소스 서버로 사용

- TF-Serving, KF-Serving, NVidia Triton, ..

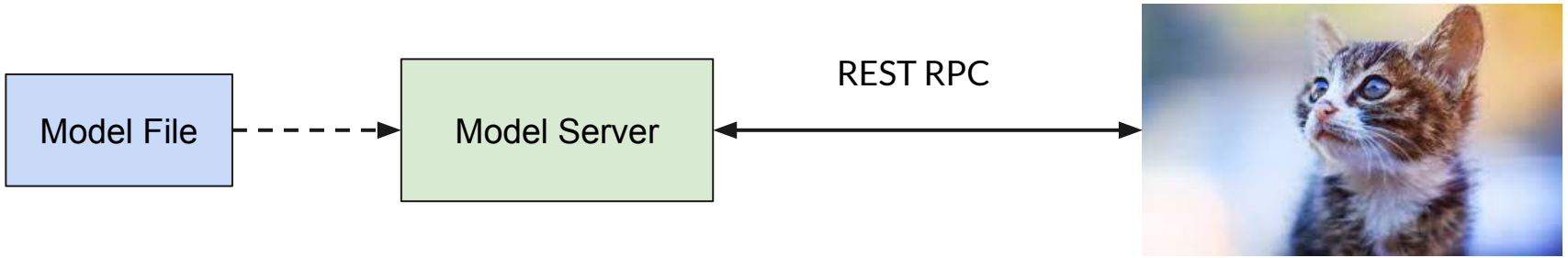
VM 위에 오픈소스 서버들 사용

- 오픈소스 서버들 (TF Serving)등이 워크플로우로 제공.

Model Servers

모델 서버란?

- ML모델들을 scalable하게 배포(deploy) 하도록 도움.
- 트래픽 + 퍼포먼스에 따라:
  - scale,
  - performance tuning,
  - 모델 라이프사이클 관리

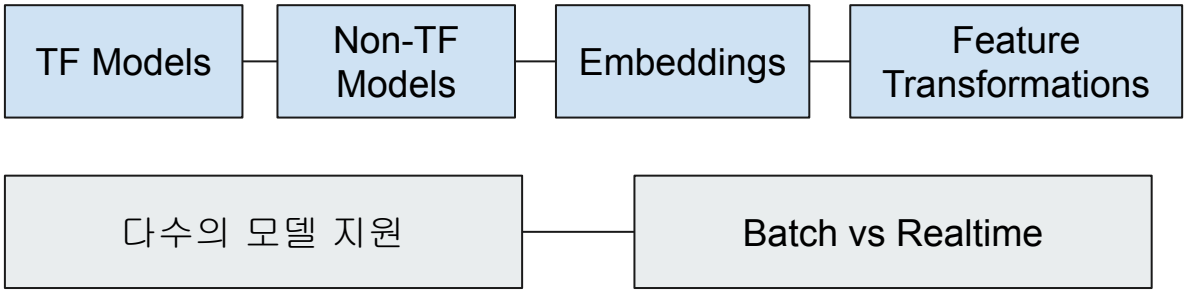


Model Server의 종류

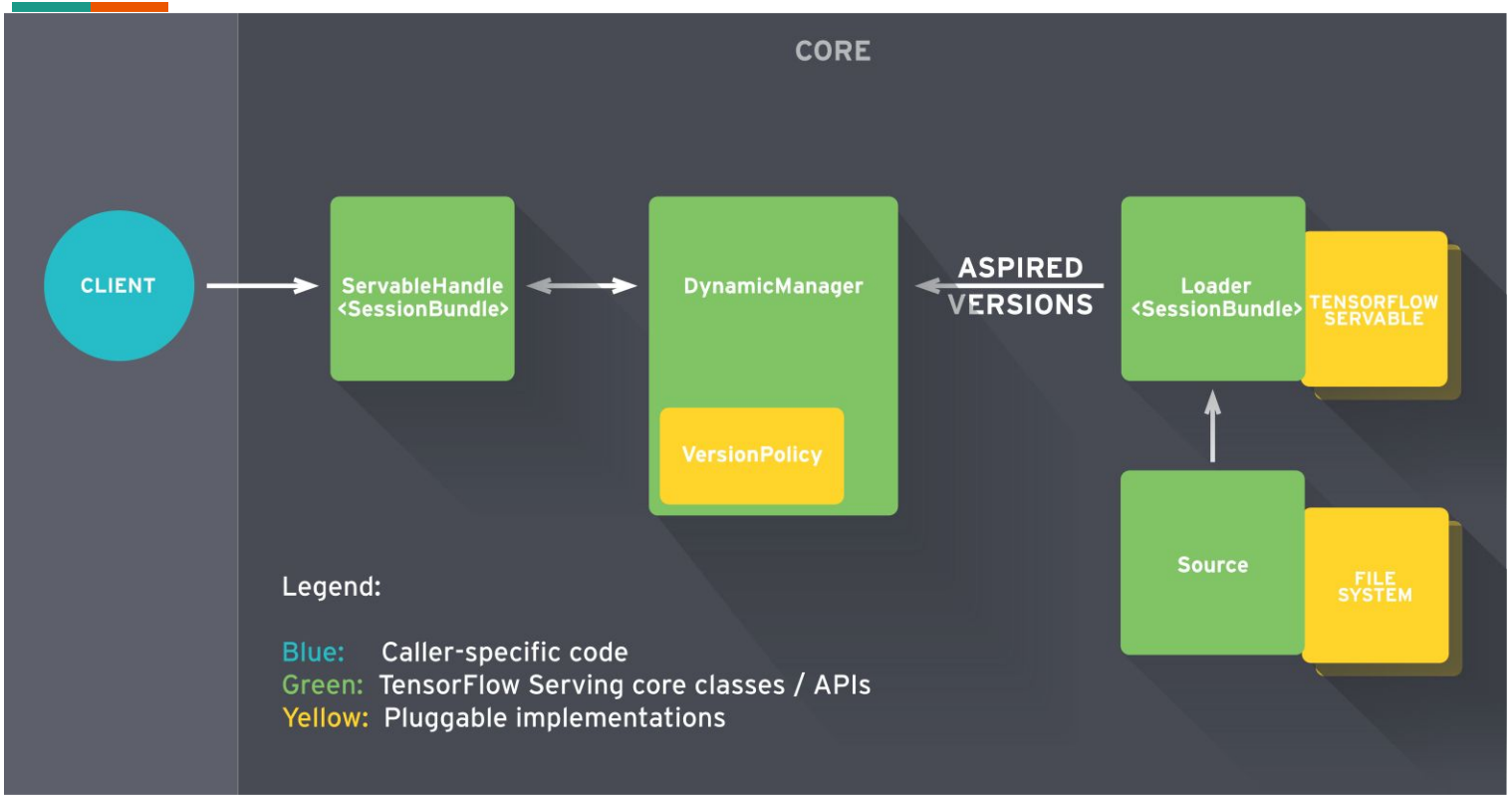
- TensorFlow **Serving**
- Pytorch **TorchServe**
- Kubeflow **KF Serving / KServe**
- Nvidia **Triton Inference Server**

TF Serving

여러 모델 관련 serving 서비스 지원



TF Serving Architecture



Scaling Infra

Scalability

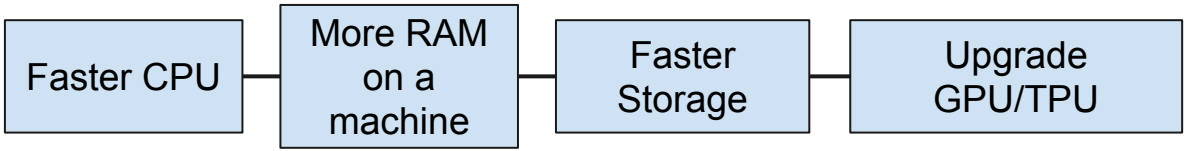
- 8강에서 보았듯이, 학습시에 어떤 task를 하느냐에 따라 CPU / GPU (TPU) / RAM / IO 등 다양한 리소스를 사용.
- 추론 때도 마찬가지로 다양한 리소스를 필요:
  - Feature transformation: CPU bound
  - Feature lookup: I/O bound
  - Inference: “Forward Pass” -> GPU/TPU

- 많은 양의 request (1M QPS)가 들어온다면?
  - Model server의 분산처리가 필요.

Vertical Scaling

Vertical Scaling:

- 시스템의 각 컴포넌트를 빠르게 변경:



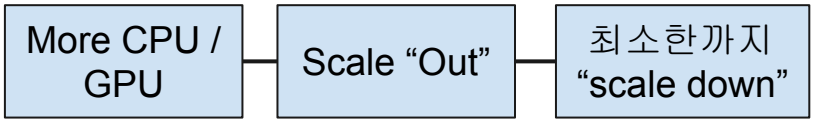
Large scale ML 시스템에 도움을 줄까?

- Somewhat..

Horizontal Scaling

Horizontal Scaling:

- 시스템을 컴포넌트별로 여러개로 늘리는 작업.



왜 scale out 했다가 줄이나?
- application이 필요로 하는 throughput까지 테스팅 후 최적화 진행.

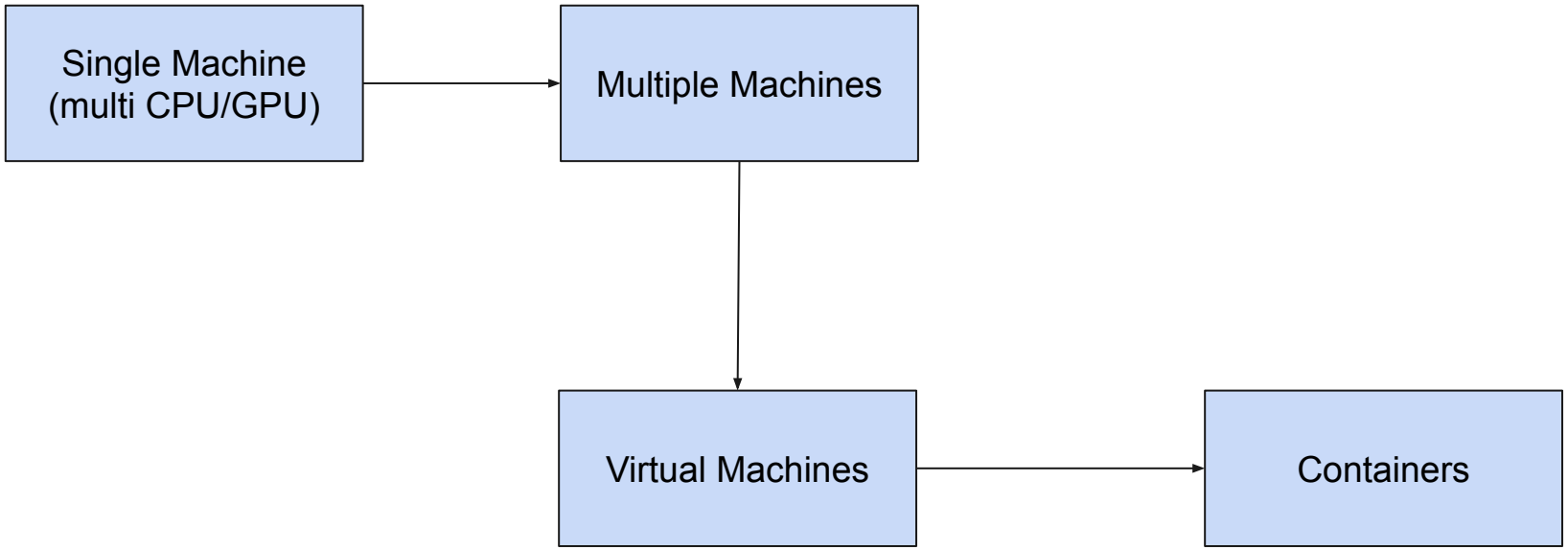
Large scale ML 시스템에 도움을 줄까?

- Yes!

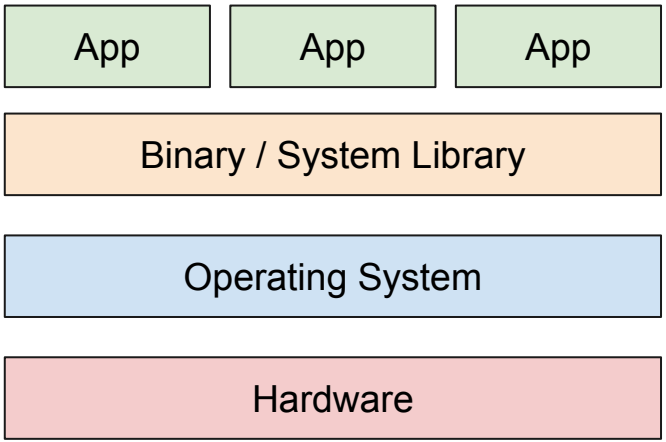
Horizontal Scaling의 장점

탄력성 (Elasticity)	트래픽 로드, latency에 따라 노드 (model server) 갯수를 늘리거나 줄임.
Uptime 유지	스케일링 작업을 위해 서버를 내릴 필요 X
무제한 하드웨어	차후 필요에 따라 하드웨어 노드를 추가 가능.

How did (horizontal) scaling change?

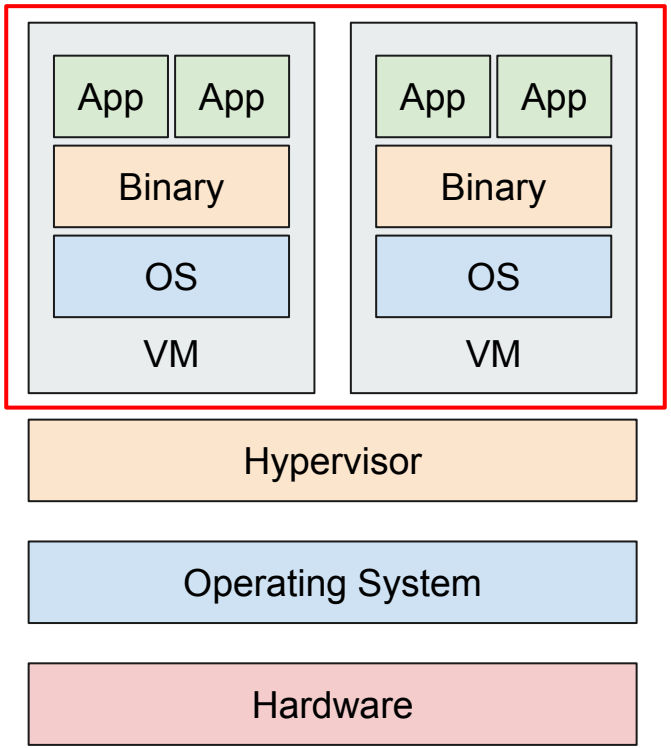


Single System Architecture



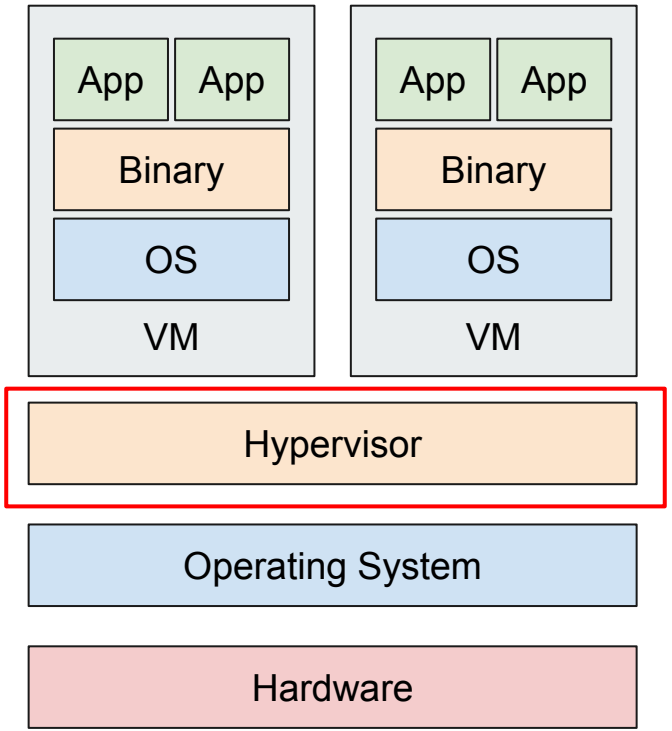
- 하나의 HW / OS에 TF.Serving 바이너리를 사용
- 하나의 바이너리를 여러 instance로 실행해 멀티 쓰레딩

Virtual Machine Architecture



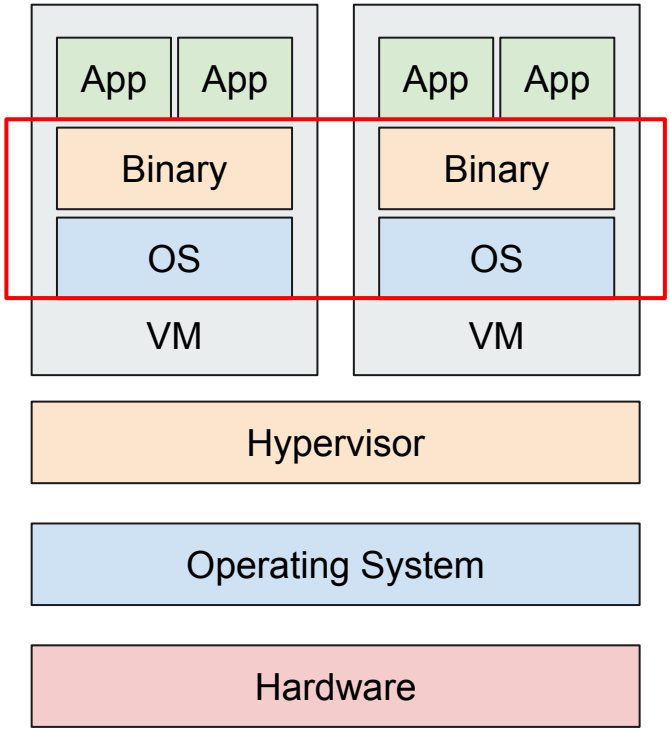
- 하나의 HW / OS위에 Hypervisor를 통해 VM을 컨트롤
  - 각 VM은 해당 OS + binary + app들로 구성.

Virtual Machine Architecture



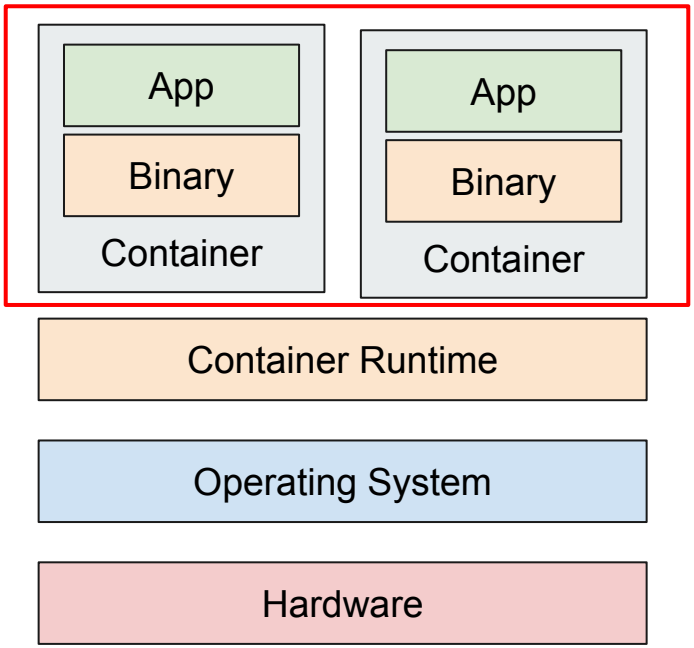
- 하나의 HW / OS위에 Hypervisor를 통해 VM을 컨트롤
- Scaling은 Hypervisor가 담당.

VM의 단점



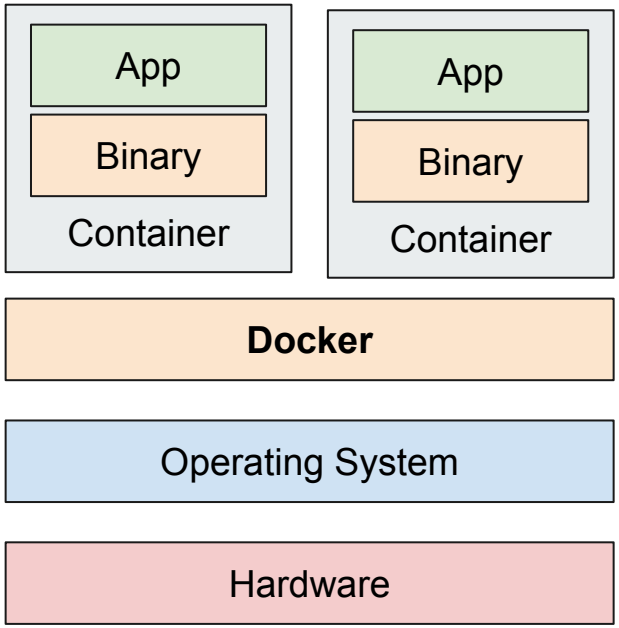
- 각 VM마다 OS가 따로 운영 → 비효율적!
  - “다른일 없이 App만 운영하는데 왜 full OS가 필요하지?”

Container Architecture



- Container Runtime이 Container들을 관리
- 각 Container마다 하나의 binary - app만 실행.
  - 비효율성을 낮춤.
- App/Binary pair로 각 ML task를 추상화.
  - deployment가 간단해짐.

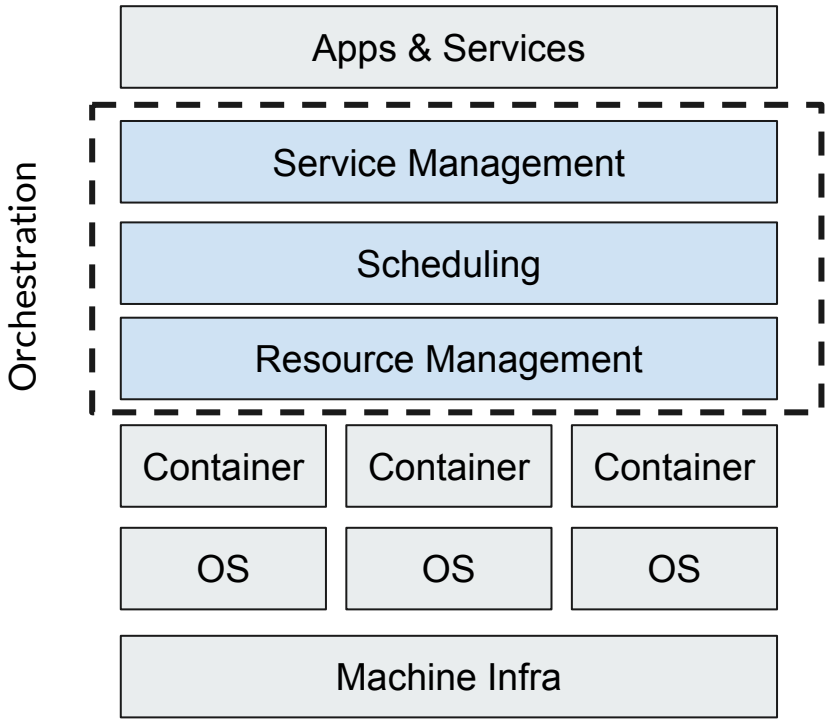
“Docker”: Container Architecture



Docker란?

- 오픈소스 컨테이너 런타임.
- 리눅스 / 윈도우 애플리케이션 지원.
- 대부분의 클라우드 서비스에서 Docker로 컨테이너화 지원.

Container Orchestration



이제 컨테이너들을 매니징 필요.

- 프로덕션 환경에서 콘테이너의 라이프사이클 관리
- 콘테이너의 스케일링 (scalability)
- 콘테이너의 신뢰성 (reliability)
- 콘테이너들 간의 리소스 분배
- 콘테이너 헬스 모니터링
- ...

Container Orchestration

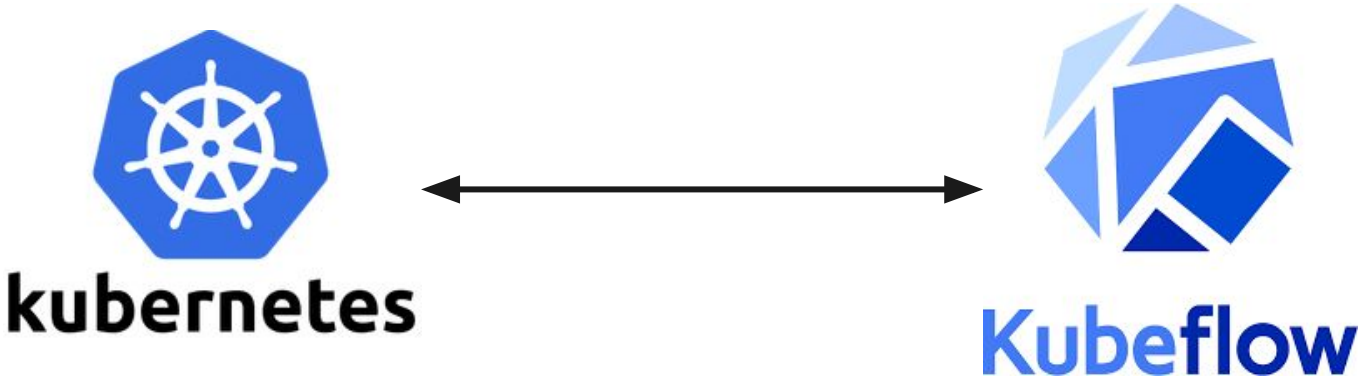
Kubernetes (쿠버네티스): 구글에서 개발한 컨테이너들을 관리하는 툴.

Docker Swarms: Docker 컨테니어들의 오케스트레이션 툴.

“ML Workflow” Container Orchestration

Kubeflow: ML workflow를 위한 컴포넌트들을 Kubernetes 위에서 관리해주는 deployment tool.

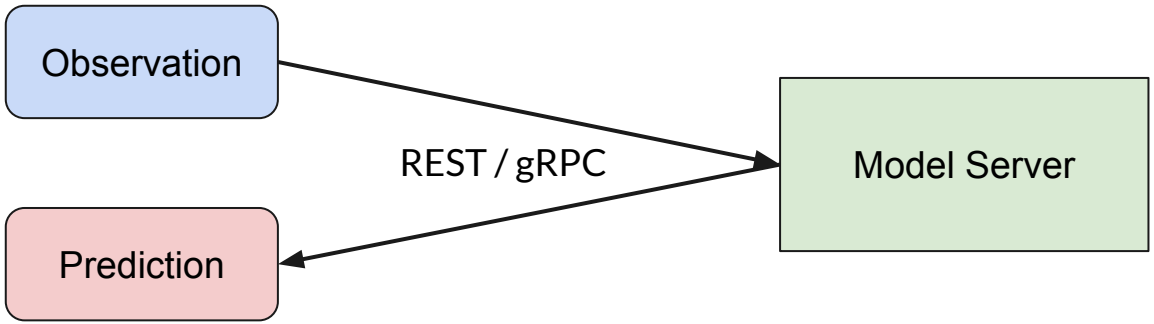
- Kubernetes에 다른 ETL, Data pipeline이 존재한다면 통합관리에 용이.
- GCP, AWS, Azure 지원.



Break

Inference Pattern

Online Inference



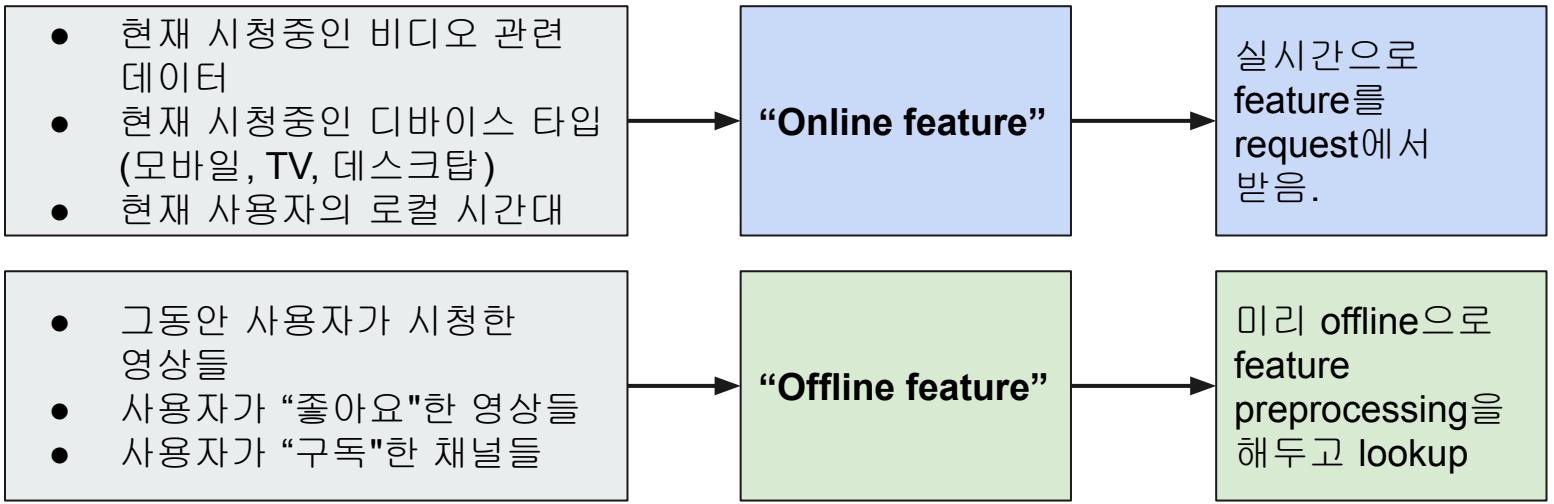
가장 "익숙한" serving pattern:

- ML prediction을 실시간으로 추론해낸다.
- Prediction이 하나의 observation으로 추론한다.
- 언제든지 리퀘스트가 들어오면 prediction을 생성해야 한다 (**on-demand**)

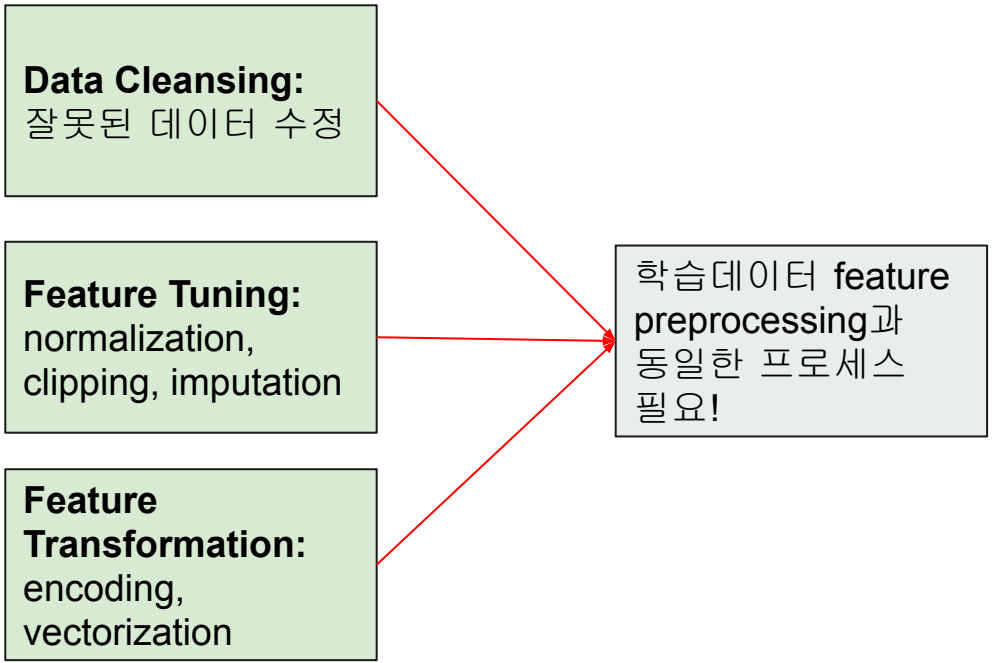
Online inference시 feature preprocessing

Online (Real-time) inference 에도, 많은 feature들은 미리 계산될 필요가 있다.

예) 유튜브에서 영상시청 시, 다음 비디오를 추천할 때:

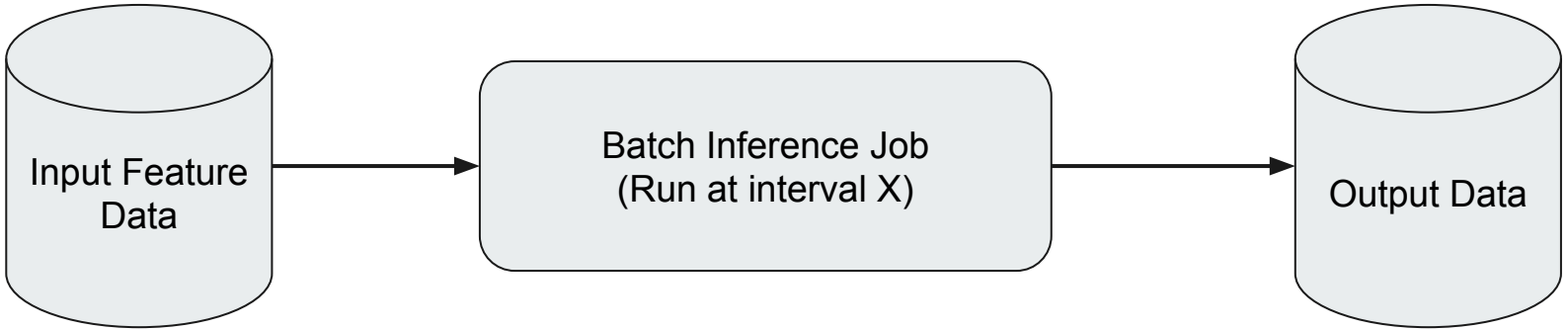


Feature Preprocessing



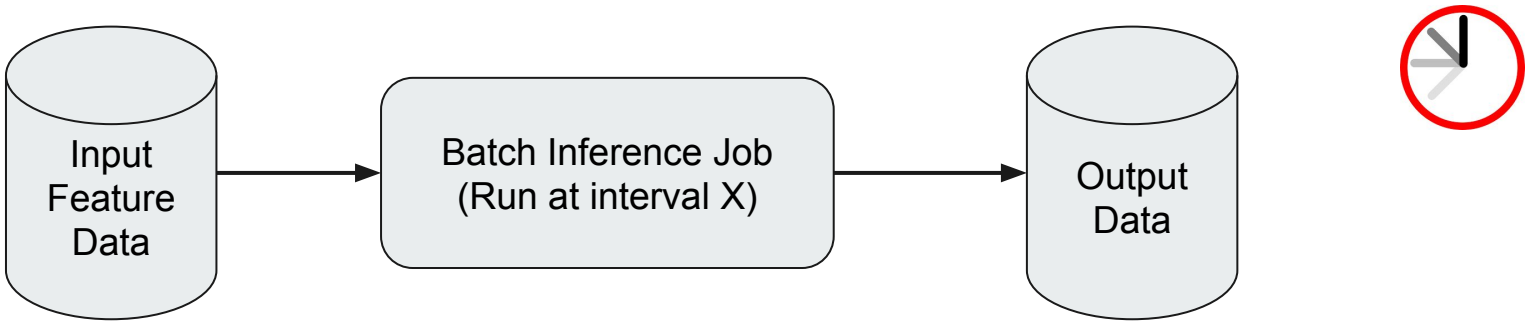
Offline Inference (Batch Inference)

- 때로는 prediction을 많은 데이터셋을 상대로 한꺼번에 배치로 추론해야 할 때가 있음.
  - **예: 매일 아침 "이 환자가 24시간 이내 퇴원할 확률은?"**
  - 하루에 한번만 추론해서 결과를 저장하는 식으로 진행.



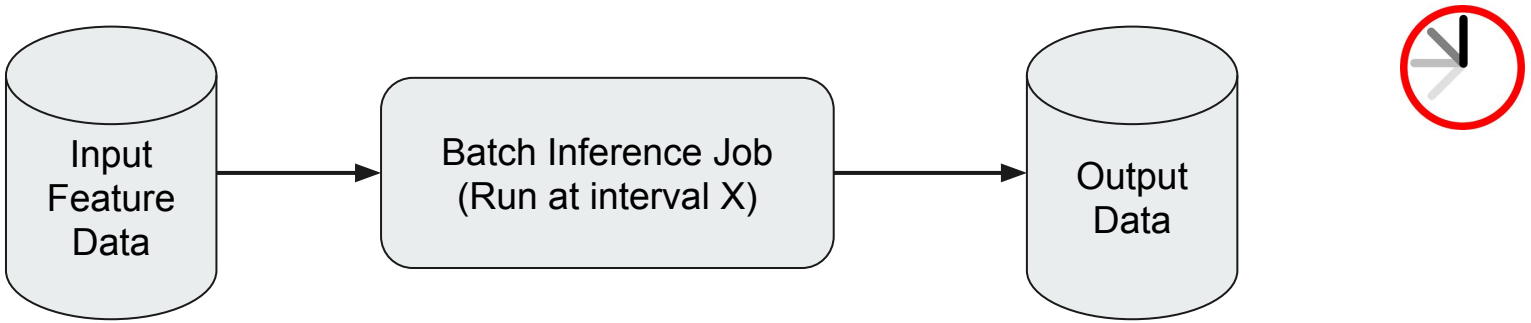
Offline Inference System의 장점

- Batch 스케줄의 인터벌이 realtime inference보다 김:
  - ⇨ Per-inference latency에 보다 자유로움.
  - ⇨ 모델 복잡도를 높여서 더 높은 정확도를 얻을 수 있음.
- Caching storage가 필요 X:
  - ⇨ 모든 feature 들을 그때 그때 cleaning, validation, transformation 할 수 있음.



Offline Inference System의 단점

- 자주 inference를 하지 않음.
  - ⇨ 오래된 데이터를 기반으로 prediction 생성.
  - ⇨ 새로운 데이터를 써야 할 때 output 생성까지 긴 interval.



Offline Inference의 예: (생각보다 많음!)

E-commerce 쇼핑:

- 쇼핑 추천 (item retrieval)시스템에서 추천할 아이템들을 주기적으로 업데이트 (batch inference)
  - 사용자의 이용패턴(로그)에 따라 추천된 아이템들을 ranking만 업데이트 할 수 있음.
- 더 복잡한 모델로 개인화 추천을 할 수 있음 (단, 딜레이 있는 데이터로).
  - 사용자가 해당 웹사이트와 interaction하는 frequency가 24시간중 잦지 않다면 24시간마다 inference해도 크게 delay되지 않을수도 있음.

유저 리뷰:

- 사용자가 서비스에 리뷰를 남겼을 때, positive vs. negative 인지 판별.
- 이에 따라, 차후 다른 추천 모델에 적용 ➔ real time prediction이 불필요.

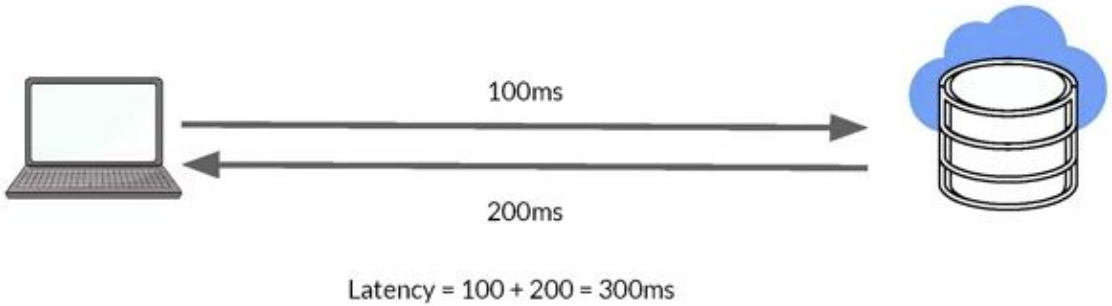
Inference Optimization

Inference Optimization

Inference를 위해 최적화할 수 있는 지표들:

1. Model Latency
2. Model Throughput
3. Cost

Latency란?



- Delay between user's action and response of application to user's action.
- Latency of the whole process, starting from sending data to server, performing inference using model and returning response.
- Minimal latency is a key requirement to maintain customer satisfaction.

Throughput이란?

- ➔ Throughput -> Number of successful requests served per unit time say one second.
- ➔ In some applications only throughput is important and not latency.

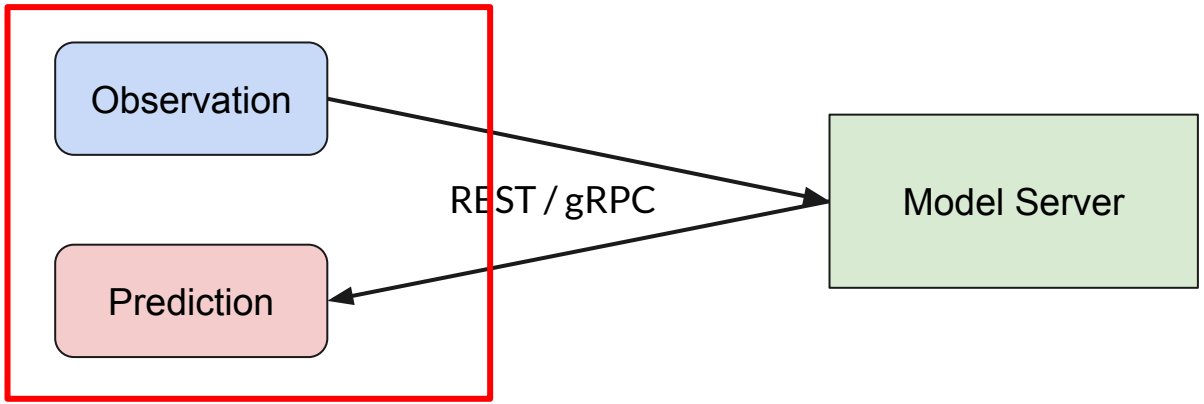
Cost는?

- The cost associated with each inference should be minimised.
  - Important Infrastructure requirements that are expensive:
    - CPU
    - Hardware Accelerators like GPU
    - Caching infrastructure for faster data retrieval.

Inference Optimization

Inference를 위해 최적화할 것들은:

1. **Model Latency**
2. Model Throughput
3. Cost

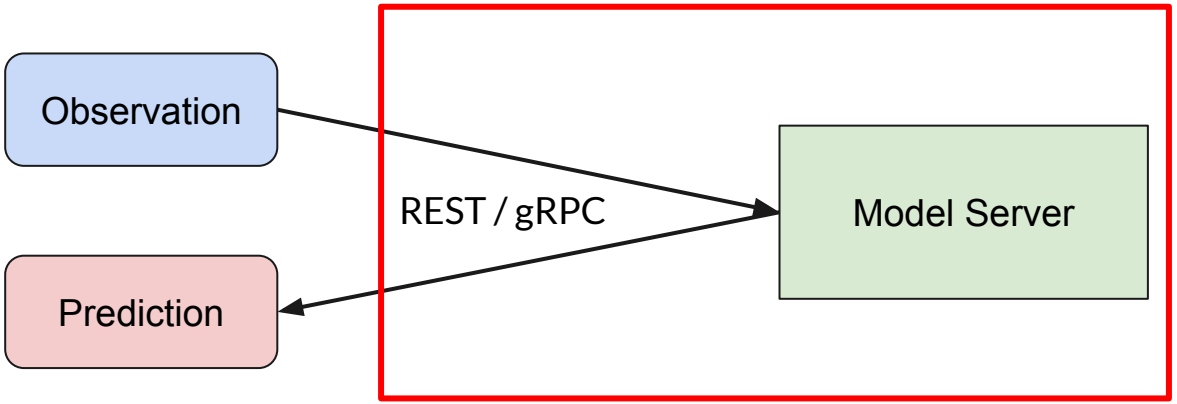


End 2 end latency: 유저의 액션 이후 해당 리퀘스트가 Observation -> Prediction이 되어 돌아오기까지의 시간

Inference Optimization

Inference를 위해 최적화할 것들은:

1. Model Latency
2. **Model Throughput**
3. Cost

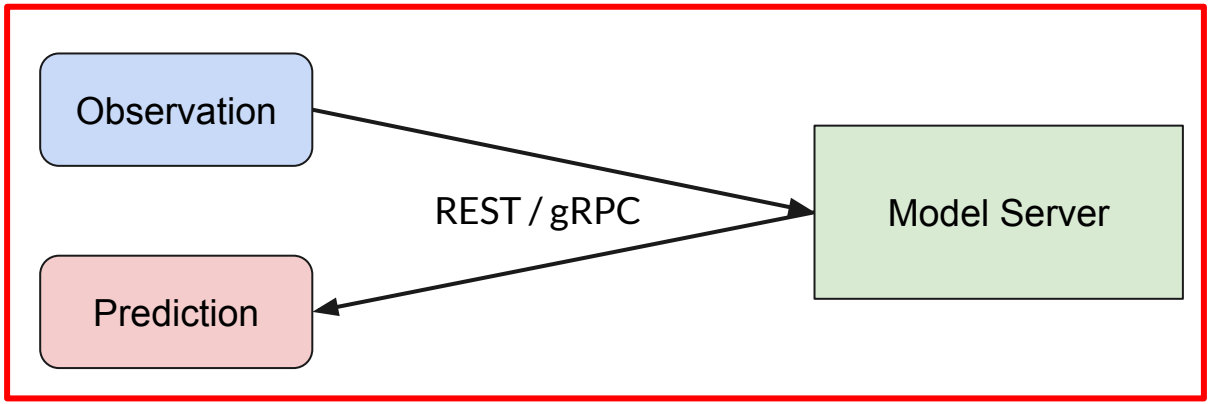


Throughput: 주어진 시간 안에 얼마나 많은 리퀘스트를 핸들링 할수 있는지 (QPS)

Inference Optimization

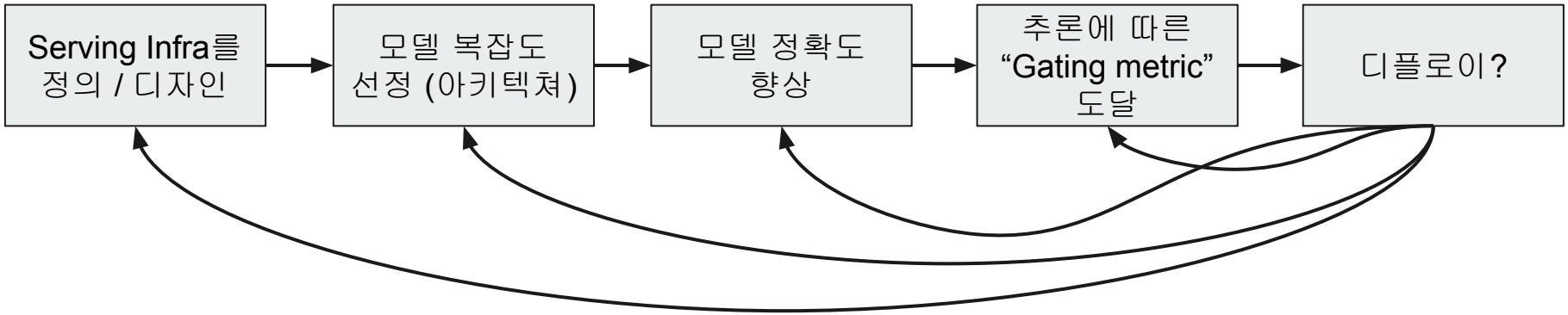
Inference를 위해 최적화할 것들은:

1. Model Latency
2. Model Throughput
3. **Cost**



Cost: 시스템 전반적으로 운영하는데 들어가는 cost.
(server / cloud service / ...)

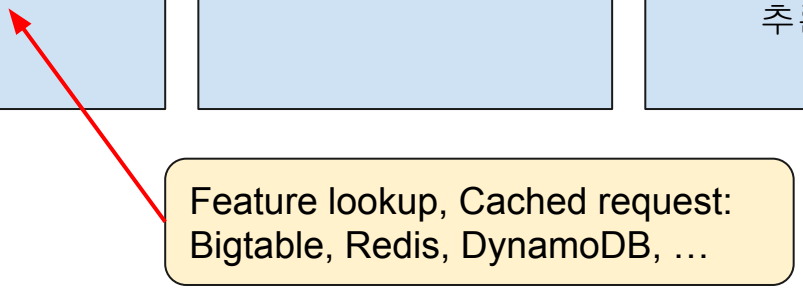
Steps to Inference Optimization



Gating metric: 어플리케이션이 요구하는 latency, throughput 등을 넘어섰는가?

Inference Optimization Techniques

Infra 최적화	**Model 아키텍쳐 최적화**	**Model Compilation**
• 트래픽에 따른 Container scale up&down • **Fast I/O storage**	• Quantization • Sparsity • SVD • Compression	• GPU/TPU 를 위해 model을 컴파일 (Model code + Interpreter가 아닌 바이너리로 추론 가능.)



Side note: Fast Lookup (Cache)

Caching lookup이나 Feature lookup 테이블은 **빠른 I/O 스토리지 시스템**을 사용:

NoSQL Databases

Google Cloud Memorystore
In memory cache, sub-millisecond read latency

Google Cloud Firestore
Scaleable, can handle slowly changing data, millisecond read latency

Google Cloud Bigtable
Scaleable, handles dynamically changing data, millisecond read latency

Amazon DynamoDB
Single digit millisecond read latency, in memory cache available

Expensive.
Carefully choose caching requirements

Optimization for Online vs Offline

Online (Real time)

- **Latency**를 최적화 필요: User experience
- Model optimization, Caching lookup (precompute)

Offline (Batch)

- **Throughput**을 최적화 필요: 더 많은 데이터를 한꺼번에 추론.
- GPU / TPU + 많은 worker들 사용.